Service-Oriented Robotic Architecture Supporting a Lunar Analog Test

Lorenzo Flückiger[1], Vinh To[2], and Hans Utz[3]

[1]Carnegie Mellon University, NASA Ames Research Center, Lorenzo.Flueckiger@nasa.gov
[2]PerotSystems, NASA Ames Research Center, Vinh.To@nasa.gov
[3]USRA/RIACS, NASA Ames Research Center, Hans.Utz@nasa.gov

Abstract

During the last 18 months, the Intelligent Robotics Group (IRG) of NASA Ames has transitioned its rover software from a classic ad hoc system to a Service-Oriented Robotic Architecture (SORA). Under SORA, rover controller functionalities are encapsulated as a set of services. The services interact using two distinct modalities depending on the need: remote method invocation or data distribution. The system strongly relies on middleware that offers advanced functionalities while guaranteeing robustness.

This architecture allows IRG to meet three critical space robotic systems requirements: flexibility, scalability and reliability. SORA was tested during summer 2007 at an analog lunar site: Haughton Crater (Devon Island, Canada). Two rovers were operated from a simulated habitat and remote ground control centers, allowing a full-scale evaluation of our system.

1. Introduction

Advances in robotics capabilities build on advances in computing systems. Today's robots are computing intensive systems, especially when they must operate in unstructured environments. It is no longer practical for a single person to carefully craft the entire software for a robot. Robot software systems are realized by multiple people working on a large code base, often in a distributed team. In this paper we present a Service-Oriented Robotic Architecture (SORA) addressing this need. SORA has been deployed on rovers used as an experimental platform for future lunar and planetary missions. By using "best practices" commonly used in software engineering but seldom used in robotics, we obtained a flexible, scalable and reliable software architecture that was demonstrated during a lunar analog field test.

1.1. Context

The Intelligent Robotics Group (IRG) at NASA Ames is dedicated to improved understanding of extreme environments, remote locations, and uncharted worlds. IRG conducts applied research in a wide range of areas with an emphasis on robotics system science and field testing. Current applications include planetary exploration, human-robot field work, and remote science. For instance, the "Human-Robot Site Survey" (HRSS) is a multi-year activity that is investigating techniques for lunar site survey [1]. During summer 2007, an HRSS field test took place at Haughton Crater (Devon Island, Canada) where IRG deployed two rovers on this analog lunar site above the Arctic Circle [2].

The goal of this last HRSS field test was to survey large (1 km x 1 km) areas of terrain with the rovers carrying surface and subsurface imaging instruments. The systematic survey could be coordinated from ground control or from a nearby habitat. The survey was performed mostly in an autonomous way to minimize the load on human operators and avoid astronaut sorties. The rovers used were the latest design version of the K10 series shown on Figure 1. K10s are four-wheeled vehicles designed to operate efficiently outdoors in proximity to human teams. They can drive at human walking speed (approx. 0.9 m/s) on uneven terrain and significant slopes. The current K10s are outfitted with a number of sensors: encoders, inclinometer, compass, sun-tracker, differential GPS, laser scanner and stereo-cameras, allowing them to navigate autonomously. Each K10 also has a payload of different science instruments: ground-penetrating radar [3], high-resolution lidar, and science cameras.

The versatility of the K10 rovers makes them convenient robotic platforms that can be easily adapted to different mission scenarios. However, this versatility, combined with a fast-evolving hardware design and the complexity of rover autonomy software, require a flexible software system that can be easily adapted and extended, but is still maintainable by a small team. This paper describes the software system we designed and implemented in response to this challenge.

1.2. Software for Exploration Robots

This paper introduces the SORA approach for exploration robots. SORA focuses on a *software* architecture applied to the robotic domain, and is independent from a

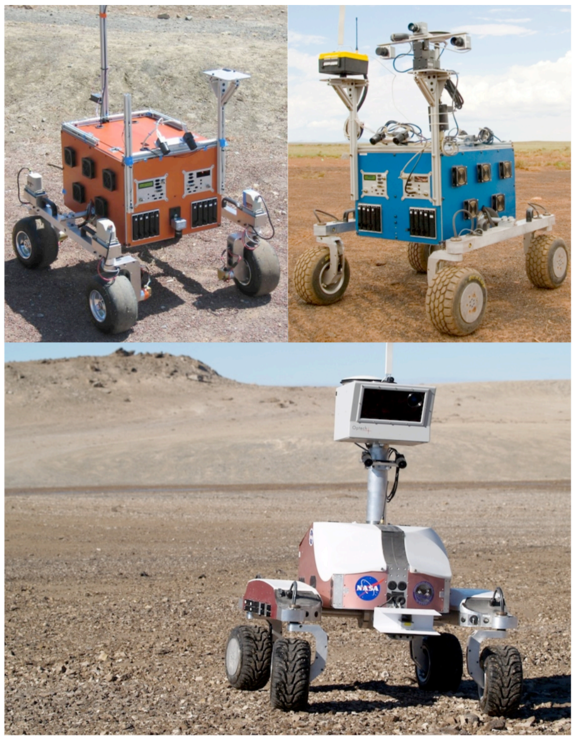

Figure 1. Evolution of the K10 rover series hardware. Initially, K10 Orange used servo motors to steer. Later, K10 blue used DC motors instead. The entire chassis was redesigned for the third generation, which includes K10 Red (pictured here at Haughton Crater).

specific *control* robotic architecture. The IRG rovers are currently using a two layer – hardware and functional – *control* architecture with an Executive responsible for plan execution. However the SORA approach does not reflect directly the structure of the robot *control* architecture. It could certainly be used with the classic 3 layer robot architecture or possibly with a behavior based architecture.

The key point of this approach, described in Section 2, is to elevate the level of abstraction by defining high level interfaces to robot services. Each rover function (locomotion, navigation, pose estimation, power, instruments, etc.) is encapsulated in a self-contained service offered to the rest of the system through a public interface. Each robot service can also consume and/or produce telemetry data using a publish/subscribe scheme. The interactions between services are conducted using the same interfaces, either internally to the rover controller (colocated function calls), or externally for remote control and monitoring (networked remote objects). The architecture relies on the use of middleware software, specifically the Common Object Request Brokering Architecture (CORBA) [4] and the Adaptive Communication Environment (ACE) [5], allowing the development of a powerful system, based on well-tested software libraries, in a short period of time.

The agility of our SORA approach, presented in Section 3, was demonstrated during the successful Haughton Crater field test, for which we were able to quickly develop brand new mission scenarios while incorporating new rover capabilities from both hardware and software points of view. This field test also allowed us to asses the robustness of the architecture and its resilience to distributed scenarios involving multiple robots managed from several locations.

The benefits of our approach consists in the flexibility given by the common interfaces exposed through services, the scalability obtained by decoupling the interactions between components, the reliability achieved thanks to the "shielding" of each service as well as the reliance on thoroughly tested middleware software.

2. SORA Approach

The robotic software developed at IRG needs to support a variety of fast evolving hardware platforms to accommodate research needs. In addition, IRG uses these robots for diverse scenarios ranging from indoor human-robot interactions to long-duration full-scale field tests in planetary analogs. Finally, the rover controller needs to integrate smoothly with other systems developed by IRG, like ground control tools and 3-D visualization systems, or by other groups like the Rover Executive [6] that can manage mission level scenarios.

In order to allow a small team working on the rover software to cope with this level of complexity while offering great flexibility and maintaining scalability, best practices in software engineering are required. In [7] we described how adopting some best practices from the Rational Unified Process (RUP) [8], usually applied to business environment, can also greatly improve the scalability and flexibility of a robotic software system in general. In this section we will present more specifically our Service-Oriented Robotic Architecture (SORA), the communication patterns used and how it benefits from existing middleware.

2.1. Service-Oriented Architecture

Our rover controller consists of a collection of services that are started on demand. There is no core controller per se, but an assembly of services that are selected by the user for a specific type of scenario. For example, a controller could consist of a single *Locomotor* service when one needs to test the rover locomotion system, or a dozen interconnected services when a full mission scenario is performed, as shown in Figure 2.

Each service provides a high level of abstraction by encapsulating a set of classes realizing a specific functionality. Services can have dependencies on other services; this is for example the case for the *Navigator* service that requires a *Locomotor* service to start and function. Some other services are totally independent and simply produce data for potential subscribers. The result is a "loosely coupled, highly cohesive system" [9], where the underlying implementation of a service can

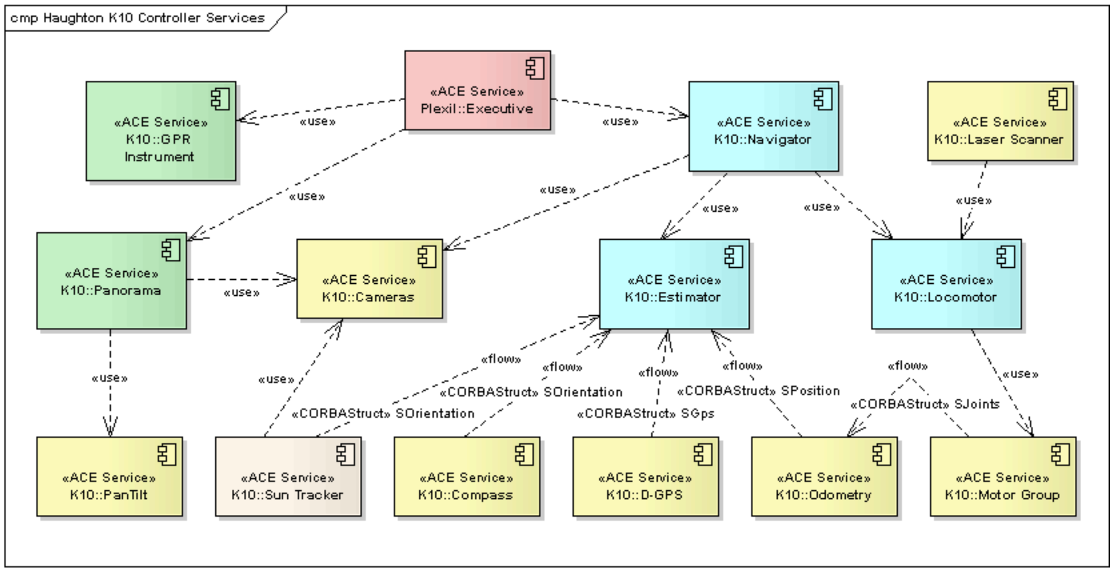
cmp Haughton K10 Controller Services
«ACE Service» Plexil::Executive
«ACE Service» K10::GPR Instrument
«ACE Service» K10::Navigator
«ACE Service» K10::Laser Scanner
«ACE Service» K10::Panorama
«ACE Service» K10::Cameras
«ACE Service» K10::Estimator
«ACE Service» K10::Locomotor
«ACE Service» K10::PanTilt
«ACE Service» K10::Sun Tracker
«ACE Service» K10::Compass
«ACE Service» K10::D-GPS
«ACE Service» K10::Odometry
«ACE Service» K10::Motor Group
«use»
«flow»
«CORBAStruct» SOrientation
«CORBAStruct» SGps
«CORBAStruct» SPosition
«CORBAStruct» SJoints

Figure 2. Services used for a K10 controller during the Haughton Crater field test. Yellow services are mostly hardware related, blue services form the core of rover mobility, green represents science instrument services and red is an external executive encapsulated into a service.

be completely changed without affecting the system as long as its public interfaces remain identical. Some of our services were recently created specifically for a mission scenario. Some other services wrap legacy components that were developed in a completely different framework.

Figure 2 lists the keys services that were used for one of the K10 controllers during the Haughton Crater field test. The services represented in yellow provide hardware abstraction. These are, for example, the motor controllers or sensors like differential GPS. These hardware abstractions are used by the mobility system composed essentially of a *Navigator* service using a *Locomotor* and *Pose Estimator* service. The *Locomotor* is responsible for the proper locomotion of the K10 four wheel drive / four wheel steer system. The *Pose Estimator* integrates information from various sensors with a Extended Kalman filter. The *Navigator* uses stereo vision to compute an optimal path to a goal while avoiding unsafe terrain. The hardware abstractions are also used by services implementing science instruments (represented in green on Figure 2). These are the only services that were different on the two K10 rovers operating at Haughton Crater, since the rovers had the same locomotion hardware but different science payload. Finally, the mission plan is controlled by an *Executive* [10]. This *Executive*, developed by another group, was encapsulated into a service, enabling it to work seamlessly with the rest of the system.

2.2. Service Details

To explain in more detail what defines one of our rover software service, we will illustrate the discussion with the *Locomotor* service represented in Figure 3. This service is responsible for the proper control and coordination of the eight motors used for the K10 locomotion. The inputs are high level motion commands expressed in the rover coordinate system. The outputs are synchronized position/velocity/acceleration trajectories for motors. The *Locomotor* service uses several "modules" (sets of classes) from the CLARAty [11] system: `motor`, `trajectory`, `locomotor_model`, `locomotor`, plus a set of supporting classes. Despite the many classes used to implement the locomotion function, the service completely abstracts this complexity for higher-level usage. The *Locomotor* service simply exports a single abstract interface to control the locomotion system and query its state.

A typical service developed in our SORA framework exhibits the following characteristics:

Exposes a public interface for control and state query. A service may export more than one interface if necessary. These interfaces are defined using the CORBA Interface Definition Language (IDL) [12] and code is automatically generated for the desired implementation languages with an IDL compiler. This allows a unique interface definition to be shared by different projects and across multiple languages. For example, the core controller is written in C++, but it interacts with a visualization tool written in Java. Figure 4 shows the interface exposed by the *Locomotor* service: it is limited to the methods that are of interest for other services or users.

Encapsulates a set of interconnected classes. The user of the service does not need to interact with a complex framework providing the desired behavior, but only with a simple clean interface. This is illustrated with

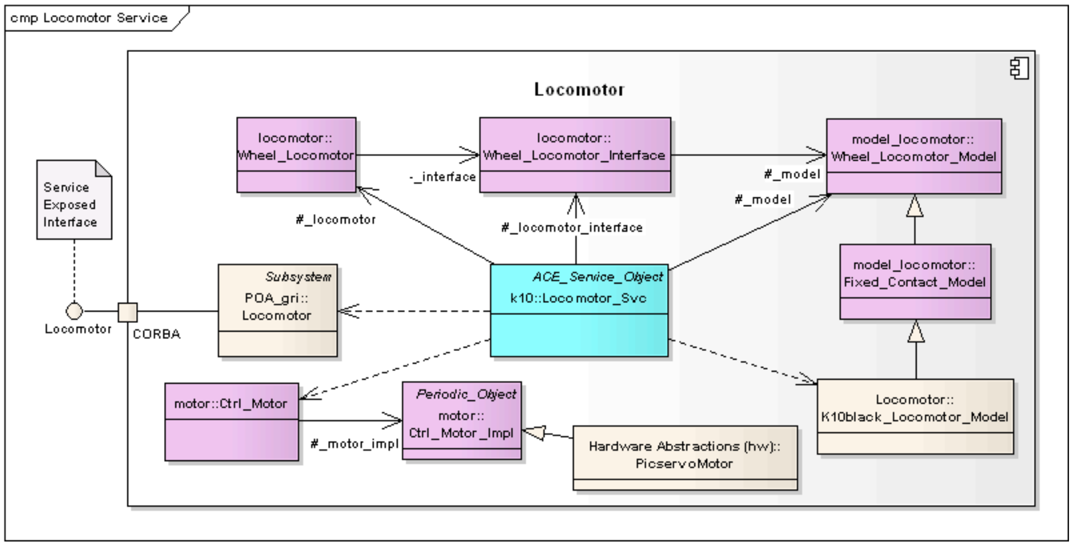

Figure 3. Details of the Locomotor *Service (only key classes are shown for clarity).*

the *Locomotor* service of Figure 3. This is a relatively simple service (compared to the navigation system for example), but already encapsulate a number of classes, mostly coming from a different framework (CLARAty). Thus the service shields the user from all the complexity associated with a large code base.

Can publish telemetry as data structures. A notification system allows sending data structures, also defined using IDL, to multiple subscribers. This is used when data distribution is more important than control by method calls. This topic will be detailed in the next Section 2.3. In our system, service state is usually accessible by either requesting information using the service interface, or by subscribing to a given class of messages. In both cases, the information is transported using the same CORBA structure.

Is self-contained and dynamically loadable. All the required dependencies (objects or other libraries) are encapsulated in the service, freeing the user of complex dependency tracking. The "Component Configurator" pattern [13] is used to combine the services in a full system. The controller uses run-time linking to load and configure individual services for a specific scenario. Our services currently are based on the `ACE_Service_Object` framework [14], which provides advanced component configuration like run-time re-configurability. The services can be discarded or re-enabled at run time, which can reduce memory footprint, and also encourage to design a system resilient to configuration changes.

2.3. Communication Patterns

The various services composing a rover controller interact using two different schemes: 1) remote object invocation when dependency between components is required, and 2) data distribution using a publish/subscribe mechanism when the components can be completely decoupled. The scheme used depends of the nature of the required interaction. The first mode is used when a service has knowledge about another service and needs to send commands to it. The second mode is used when a service is processing data generated by others, without requiring explicit knowledge of these other services. In addition, the use of a feature rich middleware like CORBA allows the communication to be unified when services are colocated (same address space) or distributed (through the network).

Remote Method Invocation Some services have dependencies on others. For example, the *Navigator* service requires a *Locomotor* service to function properly. This type of dependency is depicted with the stereotype `<<use>>` in Figure 2. The detail of the interactions between the *Navigator* and *Locomotor* is shown in Figure 5. The "ball and socket" representation clearly shows the interface `Locomotor` that is provided by *Locomotor* and required by *Navigator*. Services have the ability to discover interfaces provided by others. Once the interface with the desired identification is discovered, a service can bind to it. This binding allows the requester service to send commands to the provider using a method call on a remote object.

An additional level of decoupling is supported in our system by the systematic use of "Asynchronous Method Invocation" (AMI) [15]. This functionality is offered by CORBA, and uses the IDL compiler to generate alternate method calls for the asynchronous mode. The server side implementation (realization of a service) is greatly simplified by the use of blocking semantics. On the other hand, the client side (which calls methods of a service) greatly benefits from a non blocking call. For example, a *Navigator* command can take minutes to complete, but often a caller like the *Executive* would like to have the method return immediately to simplify its internal behavior. It will be notified by a callback when the command finally completes (or fails). All the complexity of AMI and the threading safety is handled by the middleware.

Data Distribution A second mechanism is used in our system to allow interactions between services without introducing any dependency between them. The services can exchange data using a publish/subscribe mechanism provided by the CORBA Notification Service. This mode works best when data need to be distributed at high frequency among multiple entities. This type of relationship is depicted on Figure 2 with the stereotype `<<flow>>`. This is not so much a dependency relationship as an indication of the data flow direction. Two services linked by a `<<flow>>` do not know of each other's existence; they only share the knowledge of the data structure exchanged. For example, the *Pose Estimator* service of Figure 2 subscribes to messages of type `SOrientation`. Doing so, it will receive the data published from any source of `SOrientation`, like the *Compass* or *Sun Tracker* services. There is no need to reconnectthe controller when a source of `SOrientation` is removed from—or added to—the system.

Unified Interfaces The combination of abstract interfaces between services and the features of CORBA provides the system with a powerful mechanism: from a client point of view, the method call on a public interface is exactly the same if the service is localized in the same address space or on a different computing node and accessed through the network. When the services are distributed over the network, the method calls are appropriately routed using the chosen protocol (TCP for our system). However, when the services are colocated, the method calls are optimized and do not go down to the network layer, but simply use normal function calls. The same benefit applies to data distribution, where data is either transmitted over the network when consumers and producers are on separate computing nodes, or uses local function calls to transfer data between services extremely rapidly when they are localized.

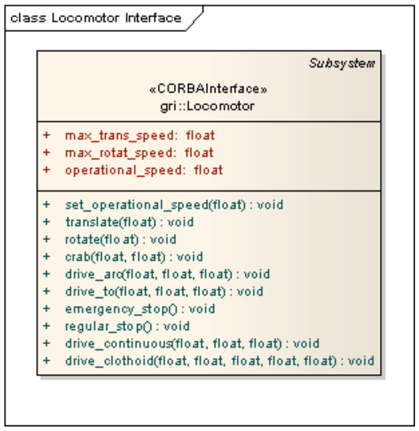
class Locomotor Interface
Subsystem
«CORBAInterface»
gri::Locomotor
+ max_trans_speed: float
+ max_rotat_speed: float
+ operational_speed: float
+ set_operational_speed(float) : void
+ translate(float) : void
+ rotate(float) : void
+ crab(float, float) : void
+ drive_arc(float, float, float) : void
+ drive_to(float, float, float) : void
+ emergency_stop() : void
+ regular_stop() : void
+ drive_continuous(float, float, float) : void
+ drive_clothoid(float, float, float, float, float) : void

Figure 4. Interface exposed by the Locomotor *Service*

This feature is shown on Figure 5 which represents a few services participating in a rover controller, `K10 Brain`, interacting with services running on two other computers: `Habitat Control` and `Remote Teleoperation`. All the services on the `K10 Brain` are localized and then communicate in a optimized way using a few function calls. The `Habitat Control` workstation is located close to the rover operation site, and benefits from high reliability and high bandwidth communication with the rover. Finally, the `Remote Teleoperation` workstation located in a remote NASA Center only has a high latency, medium bandwidth satellite link with the rover. This Lunar Analog setup was realized during our Site Survey Field test where the Habitat Control and K10 Brain were located at Devon Island (Canadian High Arctic) and the Remote Teleoperation role was played by the Johnson Space Center (JSC) in Houston, Texas.

As shown on Figure 5, the exact same interface `Locomotor` is used both by a local service *–K10 Navigator–* or by an external service *–Locomotor Control–* which represents a graphical control panel used to teleoperate the rover. The same figure also shows that the *Pose Estimator* is publishing data of the form `SPoseEstimate` which is consumed transparently by a localized service *–Navigator–* or by remote services *–habitat Viz* and *remote Viz–* which represents different instances of the same executable: a 3-D visualization and monitoring system.

These unified interfaces between services are providing the following benefits:

- flexibility of the system, since the services can be re-arranged on different nodes depending on the resources and scenarios.
- scalability, since adding more rovers and control station are done in a transparent way by adding services running on new computing nodes.
- remote inspectability, allowing unit testing during development phase and online supervision during operations.

2.4. Extensive Use of Middleware

Most of the architecture paradigms described in the previous sections would not have been realizable with such a robust and extensive set of features without the support of middleware. The communication between the services of our architecture fully relies on multiple CORBA features and services: Remote Method Invocation, Asynchronous Method Call and Notification Service for data distribution. Several features of our system, like the Component Configurator pattern, have been implemented using the Adaptive Communication Environment.

In addition, to help us bring the power of CORBA to our application, we have adopted the robotic middleware Miro [16]. Miro makes extensive use of CORBA as a communication infrastructure and customizes it for the

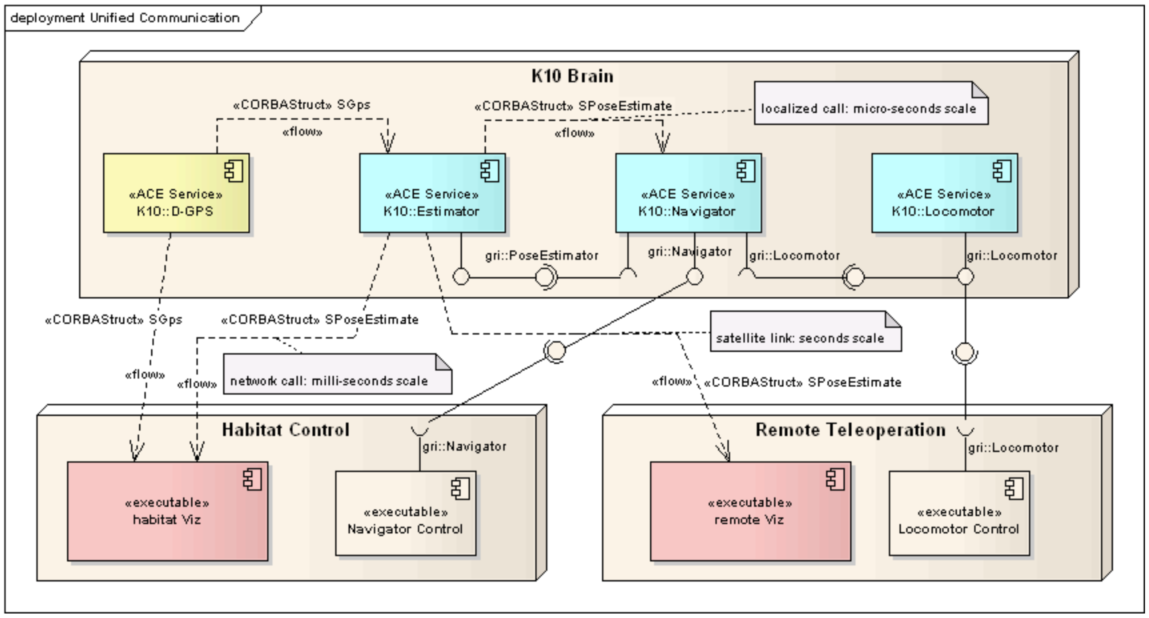
deployment Unified Communication
K10 Brain
«CORBAStruct» SGps
«flow»
«CORBAStruct» SPoseEstimate
«flow»
localized call: micro-seconds scale
«ACE Service» K10::D-GPS
«ACE Service» K10::Estimator
«ACE Service» K10::Navigator
«ACE Service» K10::Locomotor
gri::PoseEstimator
gri::Navigator
gri::Locomotor
gri::Locomotor
«CORBAStruct» SGps
«CORBAStruct» SPoseEstimate
satellite link: seconds scale
«flow»
«flow»
network call: milli-seconds scale
«flow» «CORBAStruct» SPoseEstimate
Habitat Control
gri::Navigator
Remote Teleoperation
gri::Locomotor
«executable» habitat Viz
«executable» Navigator Control
«executable» remote Viz
«executable» Locomotor Control

Figure 5. Publish/subscribe models across nodes and unified interfaces

robotic domain.

Miro offers support for the following paradigms to the robotic world:

- Distributed or colocated communication using the CORBA infrastructure.
- Publish/subscribe capabilities to distribute telemetry among components of the system using the Notifiaction Service.
- A Parameter and Configuration Management framework allowing parametric component service assembly.
- A mechanism to record all telemetry messages and replay them offline [17].

Thanks to the use of CORBA middleware, SORA would easily be transferable to robotic flight hardware providing less resources than used in K10 rovers. The CORBA specification provides extensions to support real-time communication, and minimized versions of the specification with very small footprint are available. The ACE/TAO framework used for SORA is indeed deployed in numerous telecommunication applications with embedded processors and in the aerospace and defense domain, which requires extremely reliable performance on hard real-time systems [18]. Of course, some high end algorithms used in few SORA services would need to be optimized for slower processors, but the middleware would not be the limiting factor.

3. Results

Quantitative measures to evaluate a software architecture are difficult to formulate, even more for research projects. Nonetheless we can provide results on three topics: 1) agility of the software during development, 2) pure performance, usability and reliability of the system during operations, and 3) assessments of the proposed concepts in view of the field test experience.

3.1. Software development agility

In 18 months, IRG conducted three significant field experiments with the K10 rovers. Each of this experiments had a very different mission scenario with different instrument payloads. In addition, a major hardware revision, including motors, motor controllers and kinematic parameters, was performed before each of the experiments. Finally, significant capability improvements were incorporated, like a continuous navigation system (compared to the previous stop-and-go method). Our architecture allowed all these transitions without rupture in our rover usage, by carefully identifying the interfaces to expose and progressively encapsulating rover functionalities into services.

Adoption of this architecture has greatly boosted productivity. In the 6 months preceding the Haughton Crater field test, the small team working on the rover software (about 3 people on average) was able to drastically improve the locomotion and navigation system, integrate new science instruments (GPR, LIDAR), develop ground control tools, and implement a complete mission scenario. The service architecture not only benefited the core rover team, but all parties providing software for the same field test, by providing them with a robust platform for software integration. In particular, the following tools were developed by external teams and directly interact with the rover operations:

- Viz: 3-D visualization environment monitoring mission progress and collected science data [19].

- Plexil: The rover executive managing the mission plan [10].
- A Google Earth plug-in plotting rover progress for mission outreach.
- The sun-tracker instrument ported from a previous rover framework [20].

These different programs (C++ and Java alike) are binding to one or multiple interfaces published by the rover services to obtain a seamless integration, localized or distributed over the network.

3.2. Performance during rover operations

The SORA used for the Haughton Crater field test allowed a single operator to control and monitor the progress of two rovers performing their survey. Most of the rover telemetry was shipped in real time to the habitat and displayed using the 3-D visualization tool, providing a comprehensive view of the situation. For convenience, when the two rovers were operating on different sites, or when a lot of experiments by the remote NASA ground control were conducted (thus requiring more attention), we had one rover operator assigned for each rover. Even in this situation, the system definitely allowed a small team to coordinate a full site survey carried out by two robots using several instruments.

During the field test, the two rovers drove for more than 40 km within two weeks of operations. The only controller issue we had was due to communication failure with the LIDAR (which we did not have the complete interface protocol). Otherwise, the restart of the controller during mission scenarios were due to manual interventions, for example to get out of a potentially unsafe situation during navigation. The system was also extremely resilient to communication drop-out. For example, on one site the topography of the terrain put the rover out of line-of-sight from the WiFi antenna about 30% of the time. The rover continued to operate autonomously during these periods and the visualization of the low level telemetry was resumed as soon as the rover got WiFi communication again. However, when the drop-out was longer than approximatively 10 minutes, the connection between a client and a servant would be lost. This is due to the TCP/IP protocol used to link the services, which would hit a timeout limit. In these cases, the client had to reconnect to the servant. For example with the 3-D visualization tool, this simply consist in pressing on a button `connect`, but it implies to construct clients with a re-connect functionality.

The communication performance between the services was quite satisfactory. The automatic optimization of the CORBA event distribution when localized or distributed freed the developers from having to handcraft different communication schemes. The amount of data transfered was relatively large. For example, most hardware services published telemetry at either 5 Hz or 10 Hz. The telemetry log saved on disk onboard the rover represents about 100 MB of data per hour of operation (this is not including the navigation images which represent about 6 GB/hour of raw image data). We did not encounter any limitation due to the CORBA middleware used. The external limitation obviously comes from the bandwidth available between a rover and the habitat or ground control. That is why, the telemetry stream shipped over the network was only a subset of the full data exchanged on the rover. This scheme is ideal because it avoids too much traffic on slow networks, while keeping the option of a full debugging capabilities since the rover keeps a full log of information onboard.

3.3. Validation of proposed concepts

The SORA approach proved to be very effective for the type of scenarios that will be encountered during lunar and planetary missions. The flexibility of the system allows us to quickly modify or add capabilities to the robots. The system is scalable, from one robot with a basic controller (for example only a *Locomotor* service enabling teleoperation) to multiple robots offering a large number of services. The ability to operate the robots with various modalities from different locations is also key to optimize the communication infrastructure, maximize code reuse, and minimize the support team. The following types of interactions were conducted:

- Fully autonomous rover status reporting when within communication coverage.
- Working in collaboration with a simulated astronaut conducting a sortie; the astronaut monitored the robot using a small hand-held laptop.
- Control and monitoring from the simulated planetary habitat by a minimal team.
- Teleoperation with slow communication and long time delays by a remote ground control center.

All of these scenarios worked well. Performance could be further improved using a more advanced quality-of-service feature in the Notification Service for telemetry sent over the high latency and low bandwidth link to ground control. In essence, the Notification Service does not allow the user to specify bandwidth limits for event consumers, which can easily flood a network with limited bandwidth.

The unified interfaces exposed by the active services proved to be effective down to the microsecond time scale:

- Microseconds are necessary to route data between services localized on the rover.
- Milliseconds are used when the same data goes down to the network from the rover to the habitat.
- Seconds can elapsed when shipping telemetry to remote ground control.

Again, this scalability capability is directly provided by advanced middleware.

4. Conclusion and Future work

This paper presents the Service-Oriented Robotic Architecture (SORA) developed in the context of space robotics. Its purpose is not to define another control architecture, but to support robotic software through the adoption of existing best software engineering practices. This approach provides the following benefits:

- Scalability is enabled by encapsulating functionalities into services and decoupling interactions between components. In addition, the services can be reused for various scenarios, or replaced by new versions without affecting the system.
- Flexibility is provided because a unique description of common interfaces generates multiple language bindings and automatic optimization for colocated and distributed scenarios.
- Reliability is improved because each service is well insulated from the rest of the system, and services are connected by middleware that has been extensively tested by a large community.

We demonstrated the validity of SORA during a full scale field test performed at a lunar analog site. The field test included two robots coordinated from a local habitat as well as from a remote ground station.

The agility of SORA will continue to be demonstrated by re-using and adapting the current system to a new robot IRG is introducing: *K10 Mini*, which will be a 20 kg total mass rover designed to carry few science instruments for future low cost lunar missions. In addition we plan to increase the efficiency and resilience of SORA for low quality communication links to ground control. This could be done by replicating a subset of events with lower frequency in a "low bandwidth channel" or by using an alternate data distribution service.

Beyond the SORA work applied to our own robotic experiments, we are especially interested in promoting the use of generic robotic system interfaces. We are convinced that, rather than trying to advocate the use of a particular robot architecture, there is more benefit to the robotics community in collaborating at the level of interfaces. Finally we would like to foster the use of middleware in the robotic domain. We believe that, in regard to the increasing complexity of robot software, one answer is to rely on strong software foundations developed and tested by the large software community.

Acknowledgments

This work was supported by the NASA Exploration Technology Development Program (Human-Robotic Systems project).

References

1. T. W. Fong, M. Bualat, L. Edwards, L. Flückiger, C. Kunz, S. Y. Lee, E. Park, V. To, H. Utz, N. Ackner, N. Armstrong-Crews, and J. Gannon, "Human-robot site survey and sampling for space exploration," in *AIAA Space 2006*, September 2006.
2. T. Fong, M. Allan, X. Bouyssounouse, M. G. Bualat, J. Croteau, M. C. Deans, L. Flückiger, S. Y. Lee, D. Lees, L. Keely, V. To, and H. Utz, "Robotic site survey at Haughton Crater," in *to be published, iSAIRAS*, 2008.
3. S. Kim, S. Carnes, A. Haldemann, C. Ulmer, E. Ng, and S. Arcone, "Miniature ground penetrating radar, CRUX GPR," in *Aerospace Conference*. IEEE, March 2006.
4. O. M. Group, "CORBA/IIOP specification," OMG, Framingham, MA, Tech. Rep., April 2004.
5. D. C. Schmidt, "The ADAPTIVE Communication Environment: Object-Oriented network programming components for developing client/server applications," in *Proceedings of the 12 th Annual Sun Users Group Conference*. San Francisco, CA: SUG, June 1994, pp. 214–225.
6. M. Dalal, T. Estlin, C. Fry, M. Iatauro, R. Harris, A. Jonsson, C. Pasareanu, R. Simmons, and V. Verma, "Plan exectution interchange language (PLEXIL)," NASA Technical Memorandum, November 2007.
7. L. Flückiger and H. Utz, "Lessons from applying modern software methods and technologies to robotics," in *Proceedings of the Software Development and Integration in Robotics Workshop*. Rome, Italy: ICRA, April 2007.
8. P. Kroll and P. Kruchten, *The Rational Unified Process made easy*. Addison-Wesley, 2003.
9. E. P. F. (EPF). OpenUP (unified process) web site. [Online]. Available: http://www.epfwiki.net/wikis/openup/
10. V. Verma, V. Baskaran, H. Utz, and C. Fry, "Execution, monitoring, and fault protextion demonstrated on a NASA Lunar rover testbed," in *to be published, iSAIRAS*, 2008.
11. I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin, "CLARAty and challenges of developing interoperable robotic software," in *IROS*. Las Vegas, Nevada: IEEE/RSJ, October 2003.
12. Object Management Group. (2006) OMG IDL. [Online]. Available: http://www.omg.org/gettingstarted/omg_idl.htm
13. D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. Wiley & Sons, 2000.
14. D. C. Schmidt and S. D. Huston, *C++ Network Programming*. Addison-Wesley Longman, December 2002, vol. 2.
15. D. C. Schmidt and S. Vinoski, "Programming asynchronous method invocations with CORBA messaging," *C++ Report*, vol. 11, no. 2, February 1999.
16. H. Utz, S. Sablatnög, S. Enderle, and G. K. Kraetzschmar, "Miro – middleware for mobile robot applications," *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, vol. 18, no. 4, pp. 493–497, August 2002.
17. H. Utz, G. Mayer, and G. K. Kraetzschmar, "Middleware logging facilities for experimentation and evaluation in robotics," 27th German Conference on Artificial Intelligence (KI2004), Ulm, Germany, September 2004, workshop on Methods and Technology for Empirical Evaluation of Multiagent Systems and Multirobot Teams.
18. ACE and TAO sucess stories. [Online]. Available: http://www.cs.wustl.edu/ schmidt/TAO-users.html
19. D. Lees, L. Keely, and L. Edwards, "Viz explorer - a 3-d visualization tool for planetary exploration," in *EclipseCon 2006*, Santa Clara, CA, March 2006.
20. M. C. Deans, D. Wettergreen, and D. Villa, "A sun tracker for planetary analog rovers," in *iSAIRAS*, September 2005.